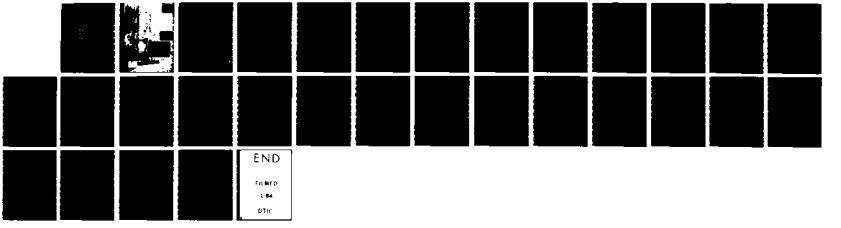
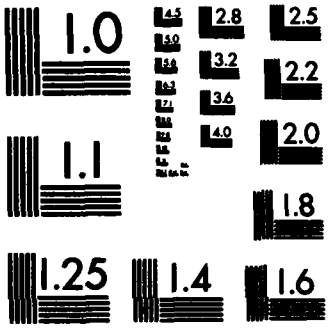
AD-A136 194 A SYSTOLIC DESIGN RULE CHECKER(U) MINNESOTA UNIV 1/1
MINNEAPOLIS DEPT OF COMPUTER SCIENCE R KANE ET AL.
JUL 83 TR-83-13 N00014-80-C-0650

UNCLASSIFIED F/G 12/1 NL





COMPUTER SCIENCE DEPARTMENT

"A Systolic Design Rule Checker

by

Rajiv Kane
Sartaj Sahni

Technical Report 83-13

July 1983

Contract N00014-80-C-0650

UNIVERSITY OF MINNESOTA

Computer Science Department

Institute of Technology

136 Lind Hall

University of Minnesota

Minneapolis, Minnesota 55455

"A Systolic Design Rule Checker

by

Rajiv Kane
Sartaj Sahni

Technical Report 83-13

July 1983

Contract N00014-80-C-0650

A Systolic Design Rule Checker*

Rajiv Kane and Sartaj Sahni
University of Minnesota

Abstract

The authors
~~We~~ develop a systolic design rule checker (SDRC) for rectilinear geometries. This SDRC reports all width and spacing violations. It is expected to result in a significant speed up of the design rule check phase of chip design.

Keywords and Phrases

Design Rule Checks, feature width, spacing, rectilinear geometries, systolic systems.

*This research was supported in part by the Office of Naval Research under contract N00014-80-C-0650 and in part by Microelectronics and Information Sciences Center at the University of Minnesota.

1. Introduction

Rapid advances in technology are making it possible to fabricate circuits of an ever increasing complexity. This increase in circuit complexity poses a severe challenge to the algorithms presently in use in design automation tools. One of the ways to meet the challenge is to develop new computer architechures capable of running these design automation algorithms efficiently. Another approach is to develop yet faster algorithms.

Several new architectures and corresponding algorithms have recently been proposed for design automation. Blank et al [BLAN81] describe a bit map processor architecture suitable for boolean operations, wire routing using Lee's algorithm, and for some design rule check (DRC) functions such as shrink and expand. Mudge et al [MUDG82] describe Cytocomputer architecture adapted for DRC and Lee type wire routing. Yet another DRC architecture is described in [SEIL82]. Some other references for special purpose architectures and associated algorithms for wire routing are [DAMM82] and [NAIR82]. A parallel processing approach for logic module placement has been developed by Ueda et al [UEDA83]. Simulation has also been the focus of several new architectural studies. The most popular such development is the Yorktown Simulation Engine ([PFIS82], [DENN82], and [KRON82]). Another logic simulation machine is described by Abramovici et al [ABRA82]. In this paper, we shall be concerned with the design of a systolic system for design rule checks. Our design differs from all earlier work on special purpose architectures for design automation in that ours is the first systolic design. Of course, systolic designs have been studied for quite some time. A valuable reference is [KUNG82]. Our systolic system for DRC's differs from earlier work on hardware assisted DRC's in that it is edge based rather than bit map based. Consequently, it has the potential of being much faster than earlier designs.

Specifically, our systolic design rule checker (SDRC) checks for spacing and width errors. The design may be extended to include other design rule checks. Our design points out the potential for systolic systems in design automation applications.

2. Polygons and Errors

In arriving at our SDRC, we made several assumptions on the nature of the polygon to be handled and also on the type of errors to be checked for. First, we

assume that polygons are composed of horizontal and vertical edges only. Hence, only right angled bends are permitted. Polygons may contain holes. These holes are also restricted to be polygons with right angled bends. Figure 1 shows two example polygons that satisfy these restrictions.

This restriction on the edges composing a polygon allows a compact representation of each polygonn. This representation consist of the following :

1. *Polygon number*. Each polygon is assigned a unique number. Holes within a polygon are assigned the same number as the enclosing polygon.
2. *A sequence of polygon vertices*. This sequence begins at the lowermost left hand vertex of the polygon and is obtained by traversing the polygon so that its interior lies to the left of the edge being traversed. Since all edges are either horizontal or vertical, the polygon vertices (except the first) may be described by providing a single coordinate. Thus , the polygon of Figure 1(a) is represented as:

p, n, x_1, y_1, x_2, y_3, x_4, y_5, x_6, y_7, x_8, y_1.

The first symbol p identifies this as an enclosing polygon. n is the polygon number. In case of a hole, an h is used in place of the p. Holes are traversed such that the the interior is to the left of each edge traversed. The representation for the polygon and holes of Figure 1(b) is:

p, n, x_1, y_1, x_2, y_3, x_4, y_5, x_6, y_7, x_8, y_9, x_{10}, y_{11}, x_{12}, y_1
h, n, x_{13}, y_{13}, x_{14}, y_{15}, x_{16}, y_{17}, x_{18}, y_{19}, x_{20}, y_{13}
h, n, x_{21}, y_{21}, x_{22}, y_{23}, x_{24}, y_{25}, x_{26}, y_{21}

The SDRC assumes that the polygons are well formed. Specifically, open polygons (Figure 2(a)); polygons with shared edges (Figure 2(b)); polygon

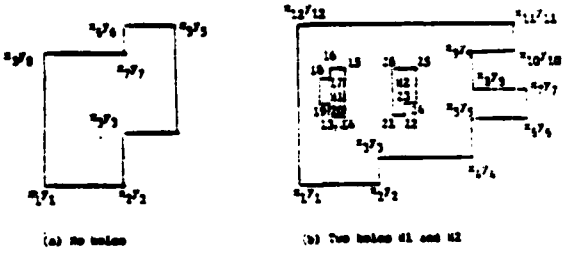


Figure 1 Examples of polygons

overlaps (Figure 2(c)); and polygons sharing an edge with a hole (Figure 2(d)) are not permitted. While this assumption of well-formedness is not essential to our disscussion, it enables us to concentrate on spacing and width issues. A minor modification to our design allows the SDRC to check for above malformations. Also, these inconsistencies need to be explicitly checked before one can apply bit map based width and spacing checks.

Let *d* denote the minimum allowable feature width. Figure 3 gives examples of polygons with width error. Note that many designers do not regard Figure 3(c) as an error unless the distance *e* is less than *d*. Our SDRC is easily changed to account for this variation. Note that The polygons of Figure 4 have no width error even though they contain some edges less than d.

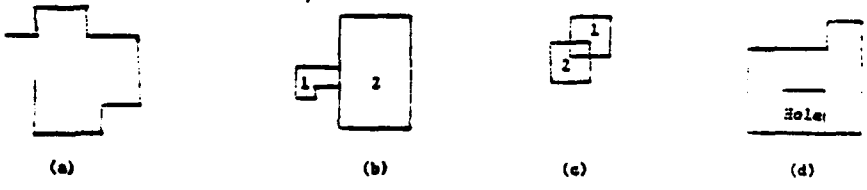


Figure 2 Malformed Polygons

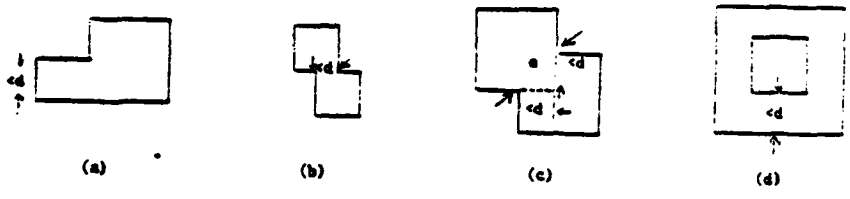


Figure 3 Polygons with width errors

Figure 4 Polygons with no width errors

Let s denote the minimum allowable spacing between polygons. The ploygons of Figure 5 have space errors at the points marked *.

As in the case of Figure 3(c), the configuration of Figure 5(c) is often not considered erroneous unless the distance labeled e is less than s. This change is also easily made in the SDRC design.

3. SDRC Architecture

The SDRC is a hardware device that may be attached to a computer system as a peripheral (Figure 6) or directly to the CPU as in case of a floating point processor.

A block diagram of the SDRC appears in Figure 7. The major components of an SDRC are two systolic sort arrays (SAX and SAY), controllers for these sort arrays, and a design rule checker (DRC). Let us assume the configuration of Figure 6. When design rule checks are to be performed, the CPU sends the compact descriptions of the polygons to the SDRC. This description is transformed into explicit edges by the controlleers for SAX and SAY. Horizontal edges are created by the cotroller for SAX and inserted into SAX. Vertical edges formed by the controller for SAY and inserted into SAY. The sort arrays sort the edges into

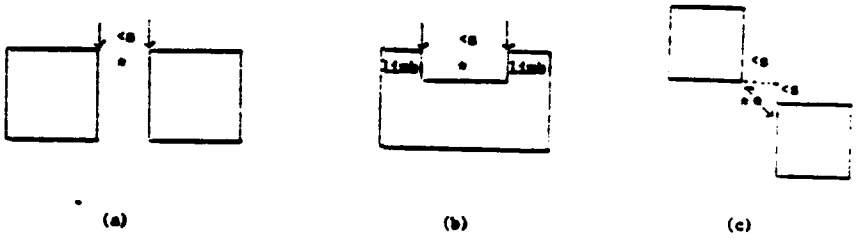


Figure 5

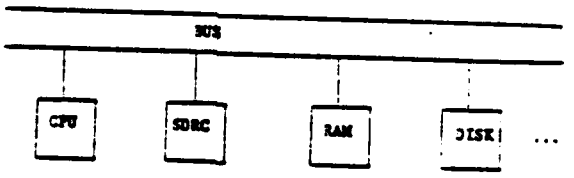


Figure 6

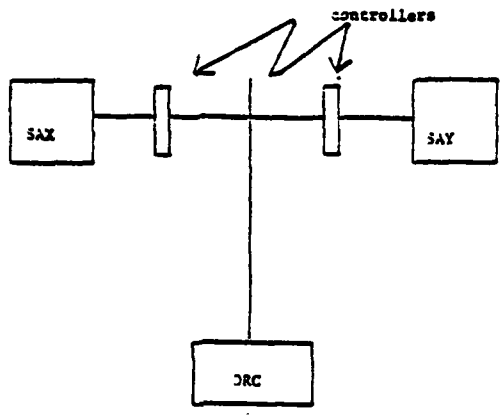


Figure 7 SDRC Architecture

lexical order. Thus, the SAX sorts edges by y - coordinates and within y - coordinate by x - coordinate. Recall that we have assumed that there are no overlapping edges. So, even though every horizontal edge has two x - coordinates, there is a unique lexical ordering for the horizontal edges. Similarly there is a unique ordering for the vertical edges.

As we shall see in the next section, the SAX and SAY are simply systolic priority queues. Consequently, as soon as the edges have been formed and entered into the SAX and SAY, they may be transmitted in lexical order to the DRC. First SAX sends its edges to the DRC, which examines them for width violations in the y direction and spacing violations in the x direction. All detected errors are transmitted back to SAX. Next SAY transmits its edges to the DRC which examines them for width errors in the x direction and spacing errors in the y direction. These errors are sent back to SAY. The errors collected in SAX and SAY may then be communicated back to the CPU.

Clearly, by using two DRCs, the horizontal and vertical edge processing may be effectively overlapped. Further, by providing a data path for the errors to go directly from the DRC to the CPU, the use of the SDRC may be pipelined.

4. Edge Forming

The descriptor for each edge formed in sort array controllers consists of 5 fields as shown in Figure 8. The terminology used in this Figure is with respect to the horizontal edges. y is the y - coordinate for the edge; x_l the left x coordinate; x_r the right coordinate; p# the polygon number; and ud (up-down) is 0 if the interior of the polygon is above this edge and 1 otherwise. In case the DRC sends errors back to the SAX (rather than directly to CPU) then each edge descriptor will have two additional bits to record the error. For vertical edges we may use the terminology of Figure 9 where x is the x coordinate of the edge; y_b and y_t are, respectively, the bottom and top y coordinates; p# is the polygon

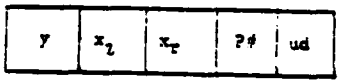


Figure 8

number; and lr (left right) is 0 if the polygon interior is to the left of the edge and is 1 otherwise. The p# field is used only to identify polygons with errors. This field may be omitted and the detected errors can be associated with polygons by performing a search at the end.

Example 1: The edge descriptors for the horizontal edges of the polygon of Figure 10 are :

$y_1, x_1, x_2, 1, 0$
$y_7, x_7, x_8, 1, 1$
$y_{16}, x_{16}, x_{15}, 1, 0$
$y_{10}, x_{10}, x_9, 1, 0$
$y_{11}, x_{11}, x_{12}, 1, 0$
$y_6, x_6, x_5, 1, 1$
$y_{14}, x_{14}, x_{13}, 1, 0$
$y_4, x_4, x_3, 1, 1$

The descriptors for the vertical edges are:

$x_2, y_2, y_3, 1, 0$
$x_8, y_8, y_6, 1, 1$
$x_{12}, y_{12}, y_{13}, 1, 1$
$x_{10}, y_{10}, y_{11}, 1, 1$
$x_{15}, y_{15}, y_{14}, 1, 0$
$x_5, y_5, y_4, 1, 1$

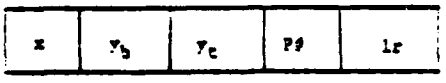


Figure 9

$x_7, y_7, y_{16}, 1, 1$
$x_1, y_1, y_6, 1, 1$

The transformation from the compact polygon representation to the edge descriptors is relatively straightforward.

5. The Sort Arrays

While the sorting algorithms have been considered for hardware implementation ([THOM82]), priority queues appear tp be best suited for our sort application. Two systolic implementations of priority queues appear in literature. One is due to Leiserson [LEIS79], and the other due to Guibas and Liang [GUIB82]. While design of [GUIB82] is simpler than that of [LEIS79], it permits an insert/delete every four cycles as opposed to once every two cycles for the design of [LEIS79].

The systolic priority queue of [LEIS79] is a linear array of processors (PEs) each having two registers A and B (Figure 11). Each register in the priority queue is large enough to hold edge descriptor. The array of processors pulsates in regular cycles with instructions:

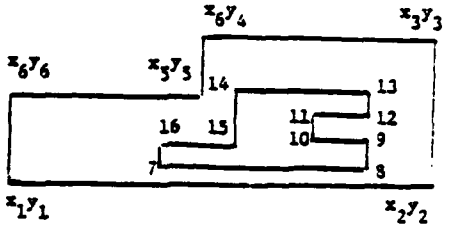


Figure 10

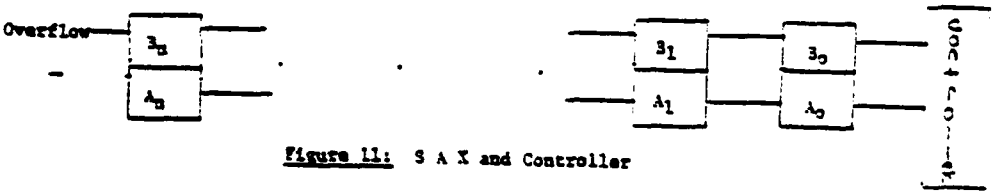


Figure 11: S A X and Controller

Figure 11

1. $B_i \leftarrow B_{i-1}$

2. Order A_{i-1}, A_i, B_{i-1} so that
$A_{i-1} \leq A_i \leq B_i$

being performed for odd i in odd cycles and for even i (i ≠0) in even cycles. A new edge can be inserted in the array just before every odd cycle by setting B_0 to the edge descriptor and A_0 to - ∞.

When all the insertions have been performed, the edges can be extracted in the lexical order by setting A_0 and B_0 to + ∞. It takes two cycles to extract each edge. The edges can be sent to DRC one by one as extracted, thereby overlapping the extraction process and DRC operation.

The remaining details for SAX and SAY may be found in [LEIS79].

The DRC

The DRC is invoked once for horizontal edges and once for vertical edges. Since the processing that occurs with horizontal edges is the same as that for vertical edges, our discussion of the DRC is confined to the case of horizontal edges.

As mensioned earlier, when processing the horizontal edges, the DRC checks for width violations in the y direction and spacing violations in the x direction. In addition, the spacing and width checks of Figure 12 are also performed.

The DRC (Figure 13) is a linear systolic array with the same organization as the priority queue of Figure 11. The A and B registers of each PE are ,however larger. In describing the fields of a register, we shall use the notation A[i].x to mean field x of register A of PE i. Each register in the DRC has all the fields

Figure 12

necessory to describe an edge(Figure 8). In addition, the following fields are also present:

PR .. This is a two bit priority field used to control the flow of data in the A and B registers. The four possible values assignable to PR have the following interpretation:

- PR = 11: This signifies an empty register. If ud = 0, then this is an empty register to the right of the rightmost edge(i.e. edge 2.1 of Figure 14) in the DRC. If ud = 1, then this is an empty register to the left of the rightmost edge in the DRC.
- PR = 10: The register contains an edge that has yet to settle in its place.
- PR = 01: This value is possible only for an A register edge. It denotes an edge that has settled.
- PR = 00: Denotes an edge for which an error has been detected.

WE .. A 1 bit width error field. It is set to 1 if a width error involving this edge has been detected.

SE .. A 1 bit space error field that is set to 1 when a spacing error involving this edge is detected.

rightok .. A 1 bit field. This is used only for edges with ud = 0. Let X, Y ∈ {A, B}. X[i].rightok = 1 iff there is a j such that
(X[i].P# = y[j].P# and X[i].x_r = Y[j].x_l and
Y[j].ud = 0)

y_{right} .. Used in conjunction with rightok. Gives the y-value of the edge that satisfies the condition of rightok

leftok .. A 1 bit field that is used only for edges with ud = 1. Let x ∈ {A, B}. X[i].leftok = 1 iff there is a limb at the leftand of the edge (Figure 5(b)).

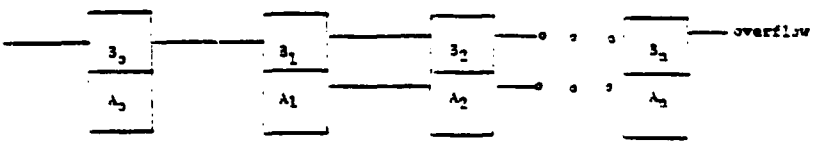


Figure 13

x_{east} .. When leftok = 1, x_{east} gives the leftmost point of the edge. Since edges may get split during processing, x_{east} may not equal x_l (x_l wil be the current left end of the split edge. Since the rightok and y_{right} fields are used only when ud = 0 while the leftok and x_{east} fields are used only when ud = 1, these fields may use the same physical register space.

It is assumed that all polygons are to be embeded on a rectangular chip (figurre 14). Thus during processing for horizontal edges, the edges 1.1, 1.2 , 2.1, and 2.2 are loaded in the SAX.The edges 1.1 and 1.2 come out of SAX before any other edges in the layout; whereas the edges 2.1, and 2.2 come out in the end. The DRC is initially loaded with the edge 2.1 for processing edges from SAX and the edge 3.1 for processing edges from the SAY.

At the start of each cycle of the DRC, an edge is inserted in B_0. This edge has PR = 01, and WE = SE = 0. Since edges come from SA (or SAY) only once every two cycles, the cycle time of the DRC must be at least twice that of the sort arrays. Once the edge enters the DRC at B_0, it moves towards the right until it finds its correct position with respect to the edges in the A registers. The A register edges are ordered by their x_l values. As the B register edges move to the right, width and spacing checks are performed against the A register edges in the PEs adjacent to the one the edge is to settle into. Once all the horizontal edges have been entered into the DRC, we set B[0].PR = 11, B[0].UD = 1 and A[0].PR = 11. This will cause the detected errors to move to the left of the DRC from where they may be removed and sent back to SAX or the CPU.

The basic cycle of the DRC is described in procedure cycle.

Before specifying the details of the step 'PROCESS_IN_EACH_PE' ,we describe a few procedures used for this purpose.

6.1 Procedures Used For Width and Spacing Checks

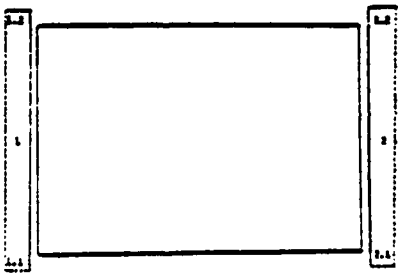


Figure 14

```
procedure cycle
  {pulsating cycle of the systolic DRC}
  repeat
    { shift B edges right }
    for every PE i, i < n do
        B[i+1] ← B[i]
    B[0] ← new edge
    B[0].leftok ← 0
    A[0].(PR,x_l,x_r, WE, SE, UD) ← (00, -∞, -∞, 0, 0, 1)
    PROCESS_IN_EACH_PE  { described later }
    { shift A edges as needed }
    for every PE i do
      if A[i].PR = A[i+1].PR = 11 and A[i+1].UD = 0
      then { mark i as right of rightmost edge }
        A[i].UD = 0
      end
      for odd i on odd cycles and
            even i on even cycles do
        if A[i].PR > A[i+1].PR
          then A[i] ←→ A[i+1] { interchange edges }
    end
  until false  { infinite loop }
end cycle
```

Spacecheck 1.1

This is used by a PE that contains an edge in its A register that is to the right of the edge in its B register. Figure 15 depicts two of the situations when the check is performed.

```
procedure spacecheck 1.1
  if A.x_l - B.x_r < s
  then [A.SE ← 1;B.SE ← 1]
end spacecheck 1.1
```

Figure 15

Spacecheck 1.2

This is similar to spacecheck 1.1 except that the B register edge is to the right of the A register edge.

```
procedure spacecheck 1.2
  if B.x_l - A.x_r < s
   then [A.SE ← 1;B.SE ← 1]
end spacecheck 1.2
```

Spacecheck2

This is used to check the interlimb distance in polygons (Figure 16). As edges progress through DRC, they may get broken. So, the edge in a register may actually be only a segment of a larger edge. The leftmost point on the original whole edge is 'remembered' in the field x_{east} which takes the place of the y_{right} (x_{east} is used when UD = 1 while y_{right} is used when UD = 0).

```
procedure spacecheck2
  if B.x_r - B.x_east < s
  then B.SE ← 1
end spacecheck2
```

Widthcheck1

This is used when the A and B register edges in a PE belong to the same polygon; have some overlap; and A.UD = 0 and B.UD = 1. Figure 17 depicts a possible situation.

```
procedure widthcheck1
  if B.y -A.y < d
```

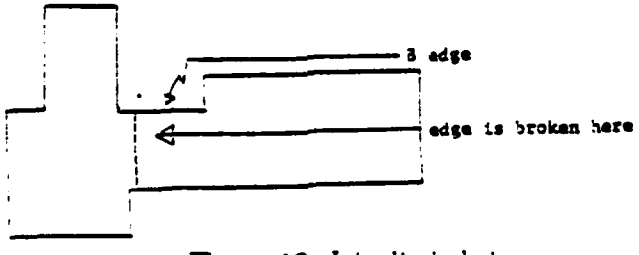


Figure 16 Interlimb distance

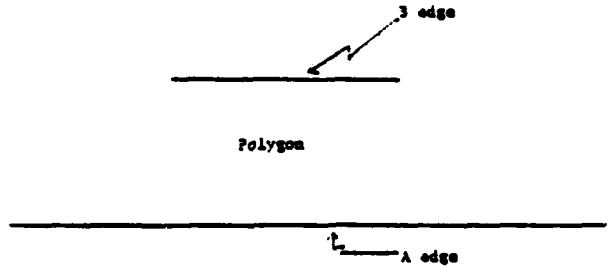


Figure 17

then [A.WE ← 1; B.WE ← 1]
end widthcheck1

Widthcheck2

The widthcheck performed by this procedure is shown in Figure 18. The PE that performs this check has edges in A and B registers that have the same polygon numbers; A.UD = 0 and B.UD = 1; and A.rightok = 1.

procedure widthcheck2
  if B.y - A.y_{right} < d
  then B.WE ← 1
end widthcheck2

6.2 PROCESS_IN_EACH_PE

In this step of the cycle, each PE examines the edges in the A and B registers and performs the checks based on this. In order to understand the edge processing procedure to be outlined shortly, it is necessary to keep the following in mind.

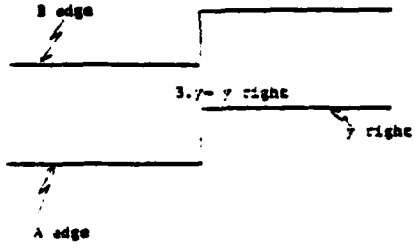


Figure 18

1. Edges may settle only in A registers. Thus,

   B.PR 01 for any PE.

2. Edges that have not yet settled must do so by moving to the right via B registers. So, the case A.PR = 10 is not possible.

3. Settled edges are ordered by their x values left to right in the A registers. The sequence of settled edges (i. e., PR = 01) may be interspersed with error edges (i. e., PR = 00) and empty edges (i. e., PR = 11).

4. A polygon edge may get split during processing. Figure 19(a) shows a polygon with a hole in it. When edge e is the B edge in the PE containing the edge acd in its A register, the acd adge is split into the three segments a, c, and d. The segments a and c are discarded. In the case of polygon in Figure 19(b), the edge e causes the edge ac to be split into segments a and c. The segment ais discarded as no new errors with respect to this segment are possible. All errors detected for the edge are retained by the segment.

In general, edge splits and discards are carried out so as to ensure that the set of active edges (i. e., PR= 01 or 10) have no overlap of their x coordinates.

The exact mechanism by which width and spacing errors are detected is best described using algorithmic notation below.

```
case A.PR of
  00 : { A edge has an error; do nothing }

  10 : { A adge hasn't settled. This is not possible. Only B edges
       may have PR = 10 }
```

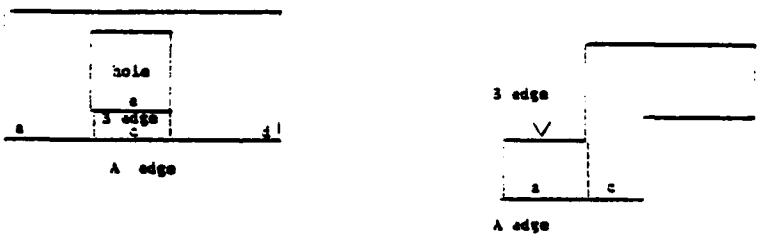


Figure 19

```
11 : { A register is empty }

   case B.PR of
     00: A ←→ B  { Move error edge to
                   empty A register }

     01: { Not possible as edges can
           settle only in A register }

     10: if A.UD = 0
         then {No edges to the right of PE }
         [B.PR ← 01; A ←→ B]
         { B edge must settle here }

     11: { do nothing }
   end case

01 : { A edge is in its correct place }

   case B.PR of
     11 : { do nothing }

     00 and 01 :{ not possible }

     10 : case A.UD of

          0:

{ At this point A.PR = 01, B.PR = 10, A.UD = 0.
  The interior of the polygon is above the edge A.}
```

{ Determine the relationship between the A and B edges }
case

1: $A.x_l \geq B.x_r$:

{ We have the situation of Figure 20 }
if B.UD = 0
then {Figure 20(a)}
[**if** $B.x_r = A.x_l$
then { By assumption on the polygons
(Figuure 2) B.p# = A.p# }
[B.rightok ← 1; $B.y_{right}$ ← A.y]
else { B.P# <> A.P# or B and A are
from two limbs of the same polygon}
spacecheck1.1
endif]
endif
{ This is B's place to settle }
A.PR ← 10; B.PR ← 01; A $\overset{\bullet}{\leftarrow\!\rightarrow}$ B
{ Notc that whcn B.UD = 1, no chccks ncod
be performed as relevant checks were
performed when the A edge was settle }

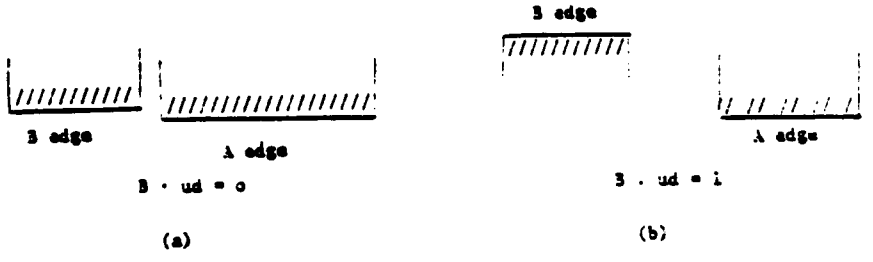


Figure 20

2: $A.x_r \leq B.x_l$:

{ This situation is depicted in Figure 21 }
if B.UD = 0
then {Figre 21(a)}
  if $A.x_r = B.x_l$
  then { By assmption on polygons (Figure 2)
      B.p# = A.p# }
    [A.rightok ← 1; $A.y_{right}$ ← B.y]
  else spacecheck1.2
  endif
else {Figure 21(b)}
  if $A.x_r = B.x_l$ **and not** B.leftok
  then { Figure 21(c). Set leftok and x_{ext}
      in case limb test is needed. B edge
      may get split later}
    [B.leftok ← 1; $B.x_{ext}$ ← $B.x_l$]
  endif
  if A.rightok
  then { Figure 21(c). A width check is needed. }
    [**if** (B.y - A.y < d) **and**
      ($B.x_l$ - $A.x_r$) < d
    then B.WE ← 1]
  endif
endif

3: **else** : { A and B edges have some overlap and so
    must be part of the same polygon. Hence
    B.UD = 1.Figure 22 - 26 show some cases.
    Note that $A.x_l \leq B.x_l < A.x_r$.

    The case $B.x_l < A.x_l$ is not
    possible as this would have caused
    caused B edge to be split earlier,
    leaving $B.x_l = A.x_l$ }

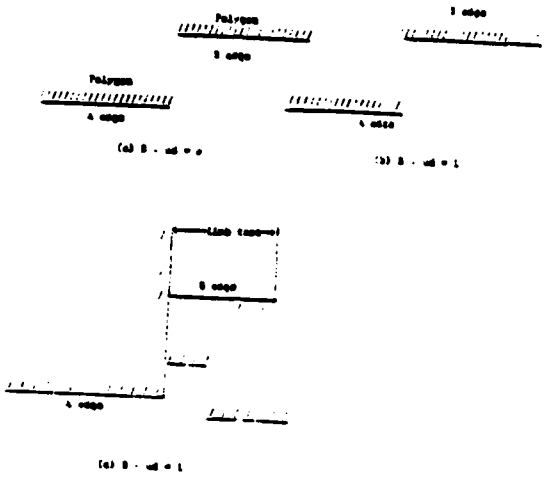


Figure 21

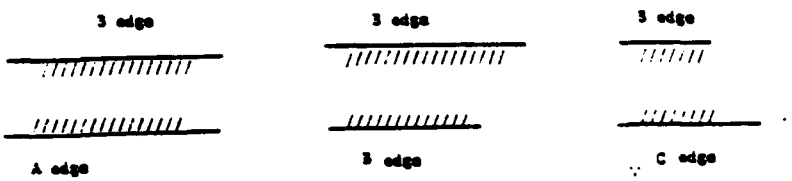


Figure 22

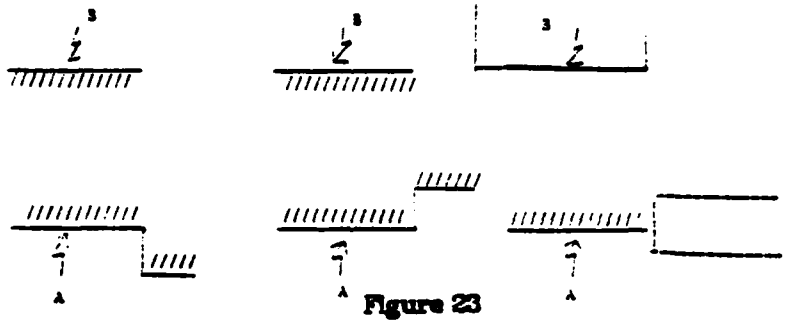


Figure 23

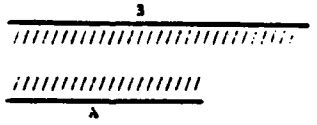


Figure 24

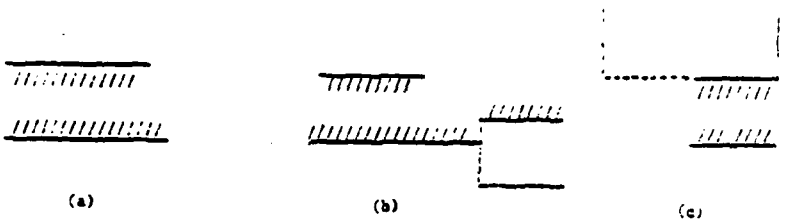


Figure 25

3.1: $A.x_l = B.x_l$: {Figure 22}

3.1.1: $A.x_r = B.x_r$:{Figure 22(a)}

if A.rightok
then { Figure 23 }
  [widthcheck2
   if B.leftok
   then { Figure 23(c) }
     spacecheck2]

{ change status of A edge }
if A.WE or A.SE
then A.PR ← 00
else [A.PR ← 11; A.UD ← 1]

3.1.2: $A.x_r < B.x_r$:{Figure 24}

{ split B edge and put left part in A;
  note that if there is a left limb of B,
  B.leftok and $B.x_{out}$ were set in case 2.
  (see Figure 21(b))}
A.UD ← 1; A.y ← B.y; $B.x_l \leftarrow A.x_r$

3.1.3: A.x_r > B.x_r : {Figure 25}

if A.rightok
then {Figure 25(b)}
  [widthcheck2]
if B.leftok
then {Figure 25(c)}
  [spacecheck2]

{ split A edge }
A.x_l ← B.x_r
{ This is B's place to settle }
A ←→ B; A.PR ← 01; B.PR ← 10

3.2: A.x_l < B.x_l : { Figure 26 - 28}

{ There is an upward limb at the left of B}
B.leftok ← 1; B.x_{est} ← B.x_l

3.2.1: A.x_r = B.x_r :{Figure 26}
  if A.rightok
  then {Figures 26(a) and (b)}
    [widthcheck2; spacecheck2]
  endif
  { split A edge }
  A.x_r ← B.x_l; A.rightok ← 0

3.2.2: $A.x_r > B.x_r$: {Figure 27(a) and (b)}

{ The situation depisted in Figure 27(c)
is not possible as the A edge would
have been split at $B.x_r$ when
edge c went over it }
if A.rightok
then { Figure 27(a)}
widthcheck2
endif
spacecheck2 { must be a limb }
{ split A edge discarding the segment
$A.x_l$ to Bx_l}
$A.x_l \leftarrow B.x_r$

3.2.3: $A.x_r < B.x_r$: {Figure 28}

{ split A and B retaining segments
b and d. (Figure 28)}
A.rightok ← 0
$(A.x_r, Bx_l) \leftarrow (B.x_l, A.x_r)$

{ This ends the case A.UD = 0}

{ Begin last case to consider }

4: A.UD = 1:

{ At this time, A.PR = 01, B.PR = 10, and A.UD = 1}

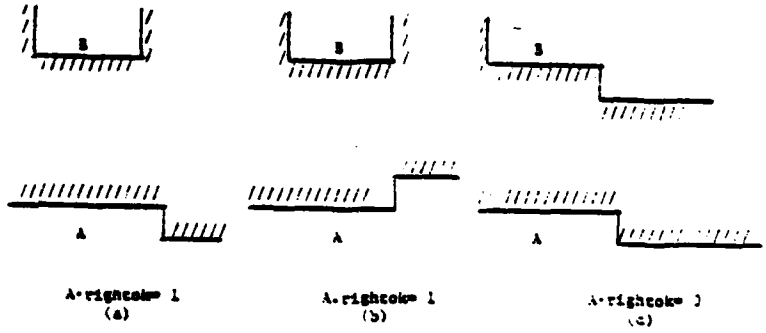


Figure 28

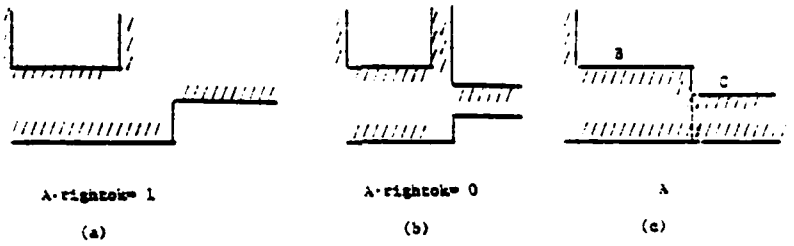


Figure 27

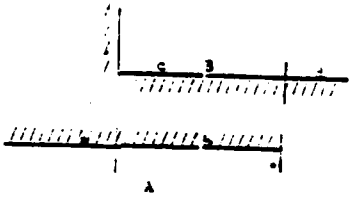


Figure 28

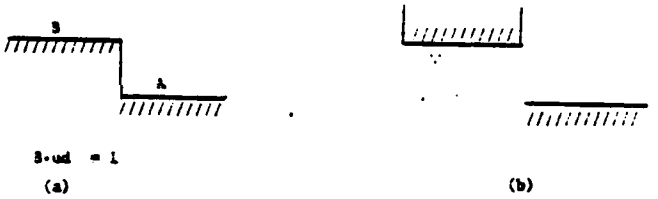


Figure 29

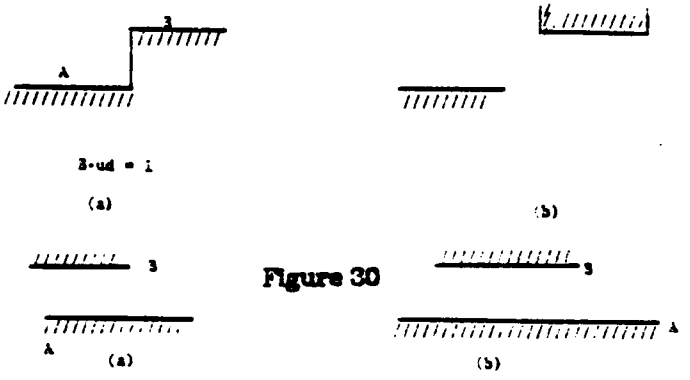


Figure 30

Figure 31

```
if B.y - A.y ≥ s

then { remaining edges are too far
       from A to cause errors}
   [if A.WE or A.SE
    then A.PR ← 00
    else A.PR = 11]
else
case

  4.1:A.x_l ≥ B.x_r: {Figure 29}
     if not [(B.UD=1 and B.x_r =A.x_l)
          or A.x_l - B.x_r ≥ s]
     then
        [A.SE ← 1; B.SE ← 1]
     { This is B's place to settle }
     A ←→ B; A.PR ← 01; B.PR ← 10

  4.2:A.x_r ≤ B.x_r : {Figure 30 }
     if not [B.UD = 1 and B.x_l = A.x_r
          or B.x_l - B.x_r ≥ s]
     then
        [A.SE ← 1; B.SE ← 1]

  4.3:else: { Partial overlap (Figure 31). So, B.UD = 0}
        A.SE ← 1; B.SE ← 1
        case

        4.3.1: B.x_r < A.x_r :{Figure 31(a) and (b) }
             { split A}
             A.x_l ← B.x_r
             A ←→ B; A.PR ← 01; B.PR ← 10

        4.3.1: B.x_r ≥ A.x_r :{Figure 31(c) and (d)}
             A.PR ← 00
```

{ The remaining spacing errors involving the left part of the A edge in Figure 31(b) and (d) will be detected when handling vertical edges }

end case

end case

end

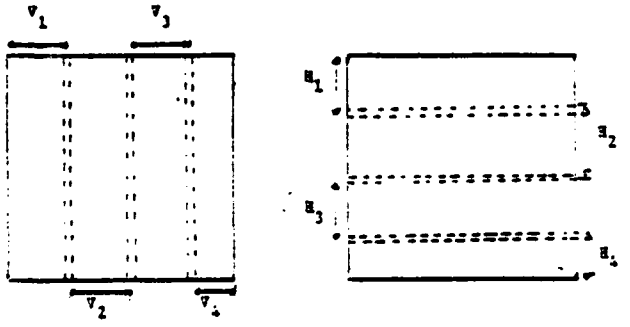
6.3 Performance

Under the assumption that the sort arrays and DRC are large enough to accommodate all the edges, the sort time and the DRC time is linear in the number of the edges in all the polygons. Furthermore the time spent extracting the errors from the sort arrays is effectively overlapped with the DRC processing.

In practice, of course, no matter how large the sort arrays and DRC, there will be times when the number of the edges to be handled will exceed the capacity of the systolic arrays. In these circumstances, the layout may be partitioned into vertical slices for SAX and horizontal slices for SAY (Figure 32). By ensuring that adjacent slices overlap by at least max(s, d) we ensure that no erroneous reporting will occur. The checks may then be performed for each slice independently.

7. Conclusions

We have demonstrated the potential of systolic architectures in the design automation field. While our design of a DRC several simplifying assumptions,



a) Partitioning into vertical slices

b) Partitioning into horizontal slices

Figure 32

these may be relaxed at the expense of the increased complexity. In particular, the assumptions about well formed polygons (Figure 1) and Manhatten vs Euclidean distance (Figure 5) are trivially removabe.

8. References

[ABRA82] M. Abramovici, Y. H. Levendel, and P. R. Menon,"A Logic Simulation Machine" *ACM IEEE Nineteenth Design Automation Conference Proceedings pp 65-73*

[BLAN81] Tom Blank, Mark Stefik, William vanCleemput "A Parallel Bit Map Processor Architecture for DA Algorithms" *ACM IEEE Eighteenth Design Automation Conference Proceedings pp 837-845*

[DENN82] M. M. Denneau, "The Yorktown Simulation Engine" *CM IEEE Nineteenth Design Automation Conference Proceedings pp 55-59*

[CUIB82] Leo J. Cuibas, Frank M. Liang, "Systolic Stacks, Queues and Counters" *1982 Conference on advanced Research in VLSI, M. I. T.*

[KRON82] E. Kronstadt and G. Pfister, "Software Support for the Yorktown Simulation Engine" *ACM IEEE Nineteenth Design Automation Conference Proceedings pp 60-64*

[KUNG82] H. T. Kung, "Let's Design Algorithms for VLSI Systems" *CMU-CS-79-151 Department of Computer Science, Carnegie Mellon Universit*

[MUDG82] T. N. Mudge, R. A. Ratenbar, R. M. Lougheed, and D. E. Atkins, "Cellular Image Processing Techniques for VLSI Circuit Layout Validation and ROuting" *ACM IEEE Nineteenth Design Automation Conference Proceedings pp 537-543*

[NAIR82] R. Nair, S. Jung, S. Liles, and R. Villani, "Global Wiring on a Wire Routing Machine" *ACM IEEE Nineteenth Design Automation Conference Proceedings pp 224-231*

[LEIS79] C. E. Leiserson, "Systolic Priority Queues" *Proceedings of Conference on VLSI: Architecture, Design, Fabrication California Institute of Technology Jan 79 pp 199-214*

[PFIS82] G. F. Pfister, "The Yorktown Simulation Engine, Introduction" *ACM IEEE Nineteenth Design Automation Conference Proceedings pp 51-54*

[SEIL82] L. Seiler, "A Hardware Assisted Design Rule Check Architecture" *ACM IEEE Nineteenth Design Automation Conference Proceedings pp 232-238*

[THOM82] C. D. Thompson, "The VLSI Complexity of Sorting" *UCB ERL M82/5 Electronics Research Laboratory, College of Engineering, Berkeley, California*

[UEDA83] Kazuhiro Ueda, Tsutomu Komatsubara and Tsutomu Hosaka, "A Parallel Processing Approach for Logic Module Placement" *ACM IEEE Transactions on Computer Aided Design Vol. CAD-2 No. 1 Jan. 83 pp 39-47*

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER TR 83-13	2. GOVT ACCESSION NO. AD-A136 194	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) "A Systolic Design Rule Checker"		5. TYPE OF REPORT & PERIOD COVERED July 1983 Technical Report 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Rajiv Kane, Sartaj Sahni		8. CONTRACT OR GRANT NUMBER(s) C-0650 N00014-80-~~C-0650~~
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Department University of Minnesota 136 Lind Hall, 207 Church Street. S.E. Mpls,MN.		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Department of the Navy Office of Naval Research Arlington, Virginia 22217		12. REPORT DATE July 1983 13. NUMBER OF PAGES 26
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Design Rule Checks, feature width, spacing rectilinear geometries, systolic systems.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

We develop a systolic design rule checker (SDRC) for rectilinear geometries. This SDRC reports all width and spacing violations. It is expected to result in a significant speed up of the design rule check phase of chip design.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

END

FILMED

1-84

DTIC